APPENDIX 9 USER PROGRAMS

1. PROGRAMS TO TRANSFER BASIC SOURCE PROGRAMS FROM COMPUTER TO COMPUTER

Here is a pair of programs which you can use to transfer information from one ATARI 800 computer to another over the telephone. These two programs demonstrate an example of a technique called "handshaking". Handshaking is a rather over used term in the computer world; what we mean here is that the receiving program will respond to the sender with an "I've got it!" message of some sort when it has successfully received each line of information from the sender.

The trick here is that the sending program must not miss the "I've got it!" message; likewise, the receiving program must not only have got the line when it says "I've got it!", but the receiver must be ready to receive the next line immediately because, theoretically, the sender might send the next line immediately. These programs show how these things are done.

Both programs operate on one line (up to 255 characters) at a time. Each program starts by DIMensioning its line-array, and each asks its user for the filename to be sent/received. Each program then opens its modem port (R1:) and disk file (assuming the send/receive files are disk files).

The SEND program is started first. In line 540, the SEND program gets a line from the disk file. The program then prints the line on the TV screen (so you can watch the data being sent). Then (lines 570-590) the line is sent over the phone. Note that port R1: is opened full duplex: SEND assumes when it receives the line, that RECEIVE might reply IMMEDIATELY. In line 600, SEND waits for the reply (In this case, a line which is empty except for an EOL is used for the reply.)

The RECEIVE routine, meanwhile, has set itself up to get a line from the modem (lines 280-290, 530). When line 530 completes (the line of data has been received), RECEIVE CLOSEs the modem port (R1:) in order to save the data on the disk (lines 540, 580) and echoes the data on the TV (line 590). Then RECEIVE OPENs the modem again and sends the reply (lines 610-630). Note that port R1: is opened full-duplex: RECEIVE assumes that it might start getting the next line IMMEDIATELY after it has sent its reply. Note also that it is not necessary for RECEIVE to INPUT the data immediately, but it is necessary that RECEIVE have started the concurrent-mode data receive (line 620).

When SEND gets the reply, it knows it can safely CLOSE the modem port (R1:) to get another line of data from its disk (lines 600-610). It then goes back to get another line of data (lines 530-540) and the whole cycle repeats. Note how the SEND program checks for the end of the disk file and how it sends a specially encoded line (EOF EOF EOF) to the RECEIVE program to signal this. Note that both programs explicitly CLOSE their files.

To use these programs, assume you and your friend are talking on the phone and you've prepared your computers (you have loaded your SEND program and your friend has the RECEIVE program). You each RUN your programs, and each program gets a filename from each of you--type the name but DON'T YET type RETURN. Now one of you sets his modem in ANSWER mode and the other sets his ORIGINATE. Looking at your watches, you decide that your friend will type his RETURN as soon as the READY light comes up on his modem and you will type your RETURN ten seconds later. In other words, the RECEIVE program must be ready to receive before the SEND program sends the first line! Now you each put your phone handsets in the modem cradles and you proceed to send a program to your friend.

Since these programs work on LINES of data, you cannot send tokenized BASIC. You should send BASIC source, that is, send a file you saved on the disk with the LIST (not SAVE) command. Your friend should ENTER the file he receives (not LOAD). You may modify these programs to send and receive the information one character at a time (using GET and PUT instead of PRINT and INPUT), doing the handshake every 40 characters or so. You'll have to pay particular attention to the question of sending the end-of-file information if you try this modification; however, such a modification should allow you to send any kind of data, not just lines of text.

Note that the RECEIVE program will probably need modification if you intend to put the received information on cassette. This is because the cassette handler requires that the first 128-byte record be written within about 30 seconds after you OPEN the cassette for output. A little experimentation should get you going.

SEND PROGRAM

```
110 DIM OUTLINE$(255)
200 REM
201 REM ==========
202 REM
210 LET TRANSLATE=32:REM [Full ATASCII]
220 XIO 38,#2,TRANSLATE,0,"R1:"
230 REM
240 PRINT "Send file's full name";
250 INPUT OUTLINE$
260 OPEN #1,4,0,OUTLINE$
500 REM
501 REM ==========
502 REM
510 FOR ETERNITY=1 TO 2 STEP 0
520 REM
530 TRAP 900:REM [Trap end file #1]
540 INPUT #1;OUTLINE$:REM [Get line]
550 PRINT OUTLINE$:REM [Echo onscreen]
560 REM
570 OPEN #2,13,0,"R1:"
580 XIO 40,#2,0,0,"R1:":REM [Start I/O]
590 PRINT #2;OUTLINE$:REM [Send line]
600 INPUT #2;OUTLINE$:REM [Get reply]
610 CLOSE #2:REM [Stop I/O]
620 REM
630 NEXT ETERNITY
900 REM
901 REM ==========
902 REM
910 OPEN #2,8,0,"R1:":REM [Send EOF]
920 PRINT #2;"EOF EOF EOF"
930 CLOSE #2:CLOSE #1:REM [All done]
999 END
```

RECEIVE PROGRAM

```
110 DIM INLINE$(255)
200 REM
201 REM ==========
202 REM
210 LET TRANSLATE=32:REM [Full ATASCII]
220 XIO 38,#1,TRANSLATE,0,"R1:"
230 REM
240 PRINT "Receive file's full name";
250 INPUT INLINE$
260 OPEN #2,8,0,INLINE$
270 REM
280 OPEN #1,13,0,"R1:"
290 XIO 40,#1,0,0,"R1:":REM [Start I/O]
500 REM
501 REM ==========
502 REM
510 FOR ETERNITY=1 TO 2 STEP 0
520 REM
530 INPUT #1;INLINE$:REM [Get line]
540 CLOSE #1:REM [Stop I/O]
550 REM
560 IF INLINE$="EOF EOF EOF" THEN 900
570 REM
580 PRINT #2;INLINE$:REM [Save line]
590 PRINT INLINE$:REM [Echo onscreen]
600 REM
610 OPEN #1,13,0,"R1:"
620 XIO 40,#1,0,0,"R1:":REM [Start I/O]
630 PRINT #1:REM [Send reply]
640 REM
650 NEXT ETERNITY
900 REM
901 REM ==========
902 REM
910 CLOSE #2:REM [EOF received]
999 END
```

2. BAUDOT TERMINAL EMULATOR

Here is a sample program showing the use of odd character transmission sizes and non-ATASCII (also non-ASCII) character codes. This program turns your ATARI computer into a BAUDOT teletype emulator.

WARNING: The ATARI 850 Interface Module was not designed for connection to old teletype equipment. Such equipment used 60 milliamp current loops rather than the more modern 20 milliamps, and high voltages [could be present] in such old equipment [which is dangerous and] could damage your 850 Interface Module. This program is intended to allow you to communicate, via a modem, over a telephone or radio link with someone owning a BAUDOT teletype.

The Baudot code is an old 5-bit serial code which is actually two codes in one. Half of the characters in Baudot are in the LETTERS SHIFT category and half are in the NUMBERS SHIFT category. The latter category includes digits 0-9 and some special characters. This program takes care of sending and receiving the shifting control characters.

This program is actually much simpler than it looks. In lines 110-210, the program's "constants" and starting values are set up. The "constants" are values which are not changed in the program, but for readability they are represented symbolically (as variables). Constants include: logical constants (YES and NO); PEEK and POKE addresses (SWITCH, KB); character constants (RETURN, FEED, UPSHIFT, DOWNSHIFT); BASIC line number constants for GOSUB's and GOTO's (RECEIVE, SEND, and TESTSWITCH); and useful numbers (NOPUSH, NOKEY). Setting INSHIFT to zero establishes LETTERS SHIFT for received data; setting ALPHA to YES establishes LETTERS SHIFT for sent data; and setting TALK to NO establishes LISTEN mode.

Lines 300-390 fill in the ASCII-BAUDOT translation tables from the data values in lines 2000-2460. REMarks are interspersed in the data to show what character is being translated. Notice that all the characters are represented within this program as numbers--the number is the "internal" character code for the corresponding letter (this is true for both ATASCII and BAUDOT, but, of course, the numbers representing a particular letter are different for each).

In order to make the code conversion easy this translation mode is set to 32--no translation. The Baud rate is set to 45.5 Baud (60 w.p.m.). This is the most common speed for old Baudot equipment. It is also the slowest speed configurable with the 850.

Lines 500-650 are the receive routine. The computer informs you that you are entering Listen mode, then OPENs the RS-232-C port R3: for input and starts the concurrent-mode input (510-540). The receive loop (560-650) first does a GOSUB TESTSWITCH to check for switching to send mode (TESTSWITCH is discussed later). The STATUS and IF PEEK... statements (580-585) see if there are any characters received. If

there are, a character is input in line 590 and translated to ASCII in lines 600-630, and PRINTed to the TV in line 640. ATASCII table values less than zero mean untranslatable characters; 0 means the LETTERS SHIFT character is received; 1 mean NUMBERS SHIFT.

Lines 700-950 are the send routine. Talk mode is announced, and port R3: is OPENed for output. The first send loop (750-950) action is a GOSUB TESTSWITCH. Line 770 checks for the typing of a keyboard key. In lines 780-800 the key's value is retrieved and its high bit is stripped (it is forced to be less than 128--this has the effect of disregarding inverse video and allows the conversion to table to require only 128 elements). The key is translated in line 810; if it translated to zero, that means it has no Baudot equivalent and line 820 restarts the loop. Otherwise, it is echoed to the TV (830); it then undergoes further translation in lines 840-890, where a LETTERS or NUMBERS shift character is added if needed. Line 900 sends the character itself, and if it was RETURN, lines 920-930 add a LINEFEED and LETTERS SHIFT.

The TESTSWITCH routine (lines 1000-1060) checks whether one of the yellow buttons is pushed (START, SELECT or OPTION). If not pushed, TESTSWITCH just RETURNs. Otherwise, the subroutine waits for the button to be released, restores BASIC's GOSUB/FOR-NEXT stack, flips from SEND to RECEIVE mode (or vice-versa) and does a GOTO to the proper routine.

In operation, the 32-character internal buffer fills with characters to be sent. When the buffer is full the Interface Module sends the characters as a block. While the characters are being sent, the keyboard will accept one character (which you won't see on the screen), so you should type the next character you want to send and wait for it to appear on the TV. Note that this program, as written, sends the block immediately when you type RETURN. You may want to experiment with variations, such as sending each character as it is typed, or reading a line at a time rather than a character at a time from the keyboard (this allows you to use backspace to correct you typing, but the person at the other end of the connection won't see anything except when you type RETURN). Have fun!

```
110 DIM ATASCII(64),BAUDOT(128)
120 REM
121 REM [Set up constants...]
122 REM ----------------------
130 LET YES=1:NO=0
140 LET SWITCH=53279:NOPUSH=7
150 LET KB=764:NOKEY=255
160 LET RETURN=8:FEED=2
170 LET UPSHIFT=27:DOWNSHIFT=31
180 LET RECEIVE=500:SEND=700
190 LET TESTSWITCH=1000
200 REM
201 REM [Starting values...]
202 REM --------------------
210 LET INSHIFT=0:ALPHA=YES:TALK=NO
300 REM
301 REM [Fill Baudot->ATASCII table...]
302 REM -------------------------------
310 FOR I=1 TO 64
320 READ IN
330 LET ATASCII(I)=IN
340 NEXT I
350 REM
351 REM [Fill ATASCII->Baudot table...]
352 REM -------------------------------
360 FOR I=1 TO 128
370 READ IN
380 LET BAUDOT(I)=IN
390 NEXT I
400 REM
401 REM [Set up I/O...]
402 REM ---------------
410 LET BAUD=128+48+1:TRANSLATE=32
420 XIO 36,#2,BAUD,0,"R3:"
430 XIO 38,#2,TRANSLATE,0,"R3:"
440 REM
450 OPEN #1,4,0,"K:"
500 REM
501 REM [Receive routine...]
502 REM --------------------
510 PRINT:PRINT "Listen..."
520 REM
530 OPEN #2,5,0,"R3:":REM [Input]
540 XIO 40,#2,0,0,"R3:":REM [Start]
550 REM
551 REM [Receive loop...]
552 REM -----------------
560 FOR INLOOP=0 TO 0 STEP 0
570 GOSUB TESTSWITCH
580 STATUS #2,PORT4
585 IF PEEK(747)=0 THEN NEXT INLOOP
590 GET #2,IN
600 LET IN=ATASCII(IN-224+INSHIFT+1)
610 IF IN<0 THEN NEXT INLOOP
```

```
620 IF IN=0 THEN INSHIFT=0:NEXT INLOOP
630 IF IN=1 THEN INSHIFT=32:NEXT INLOOP
640 PRINT CHR$(IN);
650 NEXT INLOOP
700 REM
701 REM [Send routine...]
702 REM ----------------
710 PRINT:PRINT "Talk..."
720 REM
730 OPEN #2,8,0,"R3:":REM [Output]
740 REM
741 REM [Send loop...]
742 REM -------------
750 FOR OUTLOOP=0 TO 0 STEP 0
760 GOSUB TESTSWITCH
770 IF PEEK(KB)=NOKEY THEN NEXT OUTLOOP
780 GET #1,KEY
790 LET OUT=KEY
800 IF OUT>127 THEN LET OUT=OUT-128
810 LET OUT=BAUDOT(OUT+1)
820 IF OUT=0 THEN NEXT OUTLOOP
830 PRINT CHR$(KEY)
840 IF ALPHA THEN 880
850 IF OUT<0 THEN 900
860 LET ALPHA=YES:PUT #2,DOWNSHIFT
870 GO TO 900
880 IF OUT>0 THEN 900
890 LET ALPHA=NO:PUT #2,UPSHIFT
900 PUT #2,ABS(OUT)
910 IF OUT<>RETURN THEN NEXT OUTLOOP
920 PUT #2,FEED:PUT #2,DOWNSHIFT
930 XIO 32,#2,0,0,"R3:"
940 LET ALPHA=YES
950 NEXT OUTLOOP
1000 REM
1001 REM [Listen/Talk switch test...]
1002 REM ---------------------------
1010 IF PEEK(SWITCH)=NOPUSH THEN RETURN
1020 IF PEEK(SWITCH)<>NOPUSH THEN 1020
1030 POP:POP:REM [Pop GOSUB & FOR-loop]
1040 CLOSE #2
1050 IF TALK THEN TALK=NO:GO TO RECEIVE
1060 LET TALK=YES:GO TO SEND
2000 REM
2001 REM [Baudot to ATASCII table...]
2002 REM ---------------------------
2010 REM [NUL,E,LINEFEED,A,SPACE,S,I,U]
2020 DATA -1,69,-1,65,32,83,73,85
2030 REM [RETURN,D,R,J,N,F,C,K]
2040 DATA 155,68,82,74,78,70,67,75
2050 REM [T,Z,L,W,H,Y,P,Q]
2060 DATA 84,90,76,87,72,89,80,81
2070 REM [O,B,G,Numbers,M,X,V,Letters]
2080 DATA 79,66,71,1,77,88,86,0
```

```
2090 REM [NULL,3,LF,-,SPACE,BELL,8,7]
2100 DATA -1,51,-1,45,32,253,56,55
2110 REM [RETURN,$,4,',COMMA,!,:,(]
2120 DATA 155,36,52,39,44,33,58,40
2130 REM [5,",),2,#,6,0,1]
2140 DATA 53,34,41,50,35,54,48,49
2150 REM [9,?,+,Numbers,.,/,;,Letters]
2160 DATA 57,63,43,1,46,47,59,0
2200 REM
2201 REM [ATASCII to Baudot table...]
2202 REM ----------------------------
2210 REM [Graphics characters incl. CR]
2220 DATA 0,0,0,0,0,0,0,0
2230 DATA 0,0,0,0,0,0,0,0
2240 DATA 0,0,0,0,0,0,0,0
2250 DATA 0,0,0,8,0,0,0,0
2260 REM [SPACE,!,",#,$,%,&,']
2270 DATA 4,-45,-17,-20,-9,0,-26,-11
2280 REM [(,),*,+,COMMA,-,.,/]
2290 DATA -15,-18,0,-26,-12,-3,-28,-29
2300 REM [0,1,2,3,4,5,6,7]
2310 DATA -22,-23,-19,-1,-10,-16,-21,-7
2320 REM [8,9,:,;,<,=,>,?]
2330 DATA -6,-24,-14,-30,0,0,0,-25
2340 REM [@,A,B,C,D,E,F,G]
2350 DATA 0,3,25,14,9,1,45,26
2360 REM [H,I,J,K,L,M,N,O]
2370 DATA 20,6,11,15,18,28,12,24
2380 REM [P,Q,R,S,T,U,V,W]
2390 DATA 22,23,10,5,16,7,30,19
2400 REM [X,Y,Z,Graphics characters]
2410 DATA 29,21,17,0,0,5,0,0
2420 REM [A-Z again]
2430 DATA 0,3,25,14,9,1,45,26
2440 DATA 20,6,11,15,18,28,12,24
2450 DATA 22,23,10,5,16,7,30,19
2460 DATA 29,21,17,0,0,5,0,0
9999 END
```

3. EXAMPLE OF PROGRAMMING A PRINTER THROUGH AN RS-232-C PORT

Here are two examples of programming printers connected serially through RS-232-C ports. It is assumed that there are fundamental differences between the two--the charactersitics of each printer control how that printer must be programmed. These two sample programs (or program fragments) are not intended to show general techniques, but are examples of how certain specific needs can be met.

The printer being programmed here is able to buffer and hold characters ahead of its printing (or it is so fast that it is always ready to accept characters to print). When it does not want you to send more data, it sets a READY line OFF; that line is connected here to the DSR pin on the RS-232-C port. However, the printer sets its READY line OFF early--it is still able to collect up to 32 characters after it says it's full. In other words, since the RS-232-C ports block data out in blocks of up to 32 characters, it is only necessary to monitor the DSR line once per block.

The automatic monitoring of DSR once per block is set up in line 150. In line 160, we tell the Interface Module to add LF to each CR (this printer wants the LF).

When a block is about to be sent, the Interface Module checks DSR (per our request). If it is OFF, the resulting NAK error is TRAPped (line 360), and in the TRAP routine (900 etc.) the program checks that the TRAP was really caused by the DSR being OFF. If this was the cause, the PRINT is simply retried -- eventually it will succeed because the printer will become ready again.

```
       :
       :
       :
140 OPEN #2,8,0,"R2:"
150 XIO 36,#2,0,4,"R2:":REM [Monitor DSR]
160 XIO 38,#2,64,0,"R2:":REM [Add LF to CR]
       :
       :
       :
360 TRAP 900
370 PRINT #2; ..... :REM [PRINT something to R2:]
       :
       :
       :
900 STATUS #2,PORT2:REM [Get R2: status]
910 LET READY=PEEK(746)/8:REM [Check readiness error]
920 IF INT(READY)<>INT(READY+0.5) THEN 360:REM [If so, retry]
930 REM [If here then some error other than port not ready]
       :
       :
       :
```

4. ANOTHER EXAMPLE OF PRINTER CONTROL THROUGH AN RS-232-C PORT

The printer being programmed in this example also has a READY line to signal that it is not ready to accept data. However, when it is not ready, it cannot accept any data. Therefore, the data must be sent to the printer one character at a time, checking DSR before each character. Since the PRINT statement cannot be made to send data one character at a time, we assume that the file to be printed was first written to a disk or cassette. Here is a program to read that file off the disk or cassette and print it on this printer.

The operation of this program should be fairly obvious. Once again, we assume the printer wants both CR and LF at the end of a line (lines 230-240). The file is read from disk (or tape) one character at a time. Then if the printer on port R2: is ready (540-550), the character is PUT (560). The output is then forced (FORCE SHORT BLOCK) in line 570.

```
110 DIM FILE$(16)
200 REM
201 REM ==========
202 REM
210 LET BAUD=13:REM [4800 Baud]
220 XIO 36,#2,BAUD,"R2:"
230 LET TRANSLATE=64:REM [Add LF to CR]
240 XIO 38,#2,TRANSLATE,0,"R2:"
250 REM
260 PRINT "List file's full name";
270 INPUT FILE$
280 OPEN #1,4,0,FILE$
290 REM
300 OPEN #2,8,0,"R2:"
500 REM
501 REM ==========
502 REM
510 FOR ETERNITY=0 TO 0 STEP 0
520 TRAP 900:REM [Trap end of file]
530 GET #1,CHARACTER
540 STATUS #2,XXX:REM [Check ready]
550 IF PEEK(747)<128 THEN 540
560 PUT #2,CHARACTER
570 XIO 32,#2,0,0,"R2:"
580 NEXT ETERNITY
900 REM
901 REM ==========
902 REM
910 CLOSE #2:CLOSE #1
920 END
```

5. READING A DIGITIZER: MORE INPUT THAN BASIC CAN HANDLE

This is an example of reading data from a digitizing pad. A digitizing pad is a device which is capable of sensing the position of a hand-held object (a special pen or whatever) and reporting its location to the computer.

The digitizing pad used here is capable of sending its information to the computer at speeds up to 4800 baud, so that is used here. each sampled pen position is 14 characters long: a digit indicating whether or not the button on the pen is being pushed, the x-coordinate (6 characters), the y-coordinate (6 characters), a CR and a LF. Since the LF follows the CR, the Interface Module will read it as the first character on the following input line.

If we assume that the digitizer sends the pen coordinates as fast as it can, then BASIC will not be able to keep up at 4800 Baud. A lower Baud rate might allow BASIC to get every sample, but at 300 Baud, for example, it would take about HALF A SECOND for each sample to come in (15 characters at 30 CPS)! Thus we want the data to come in at the highest possible rate. It really doesn't matter if we miss samples, because the pen is usually in pretty much the same place sample after sample.

Therefore, it is OK if the digitizer sends samples as fast as it can and the program just grabs them now and then when it can. However, we have to take into account the way the Interface Module behaves when data arrives too fast: when the computer's holding buffer fills up, the NEWEST data replaces the OLDEST. However, an INPUT statement reads the OLDEST data--which is messed up by being replaced by the newer data!

This is actually very trivial to solve. Look at line 100. A sample is INPUT twice! The first INPUT gets the messed-up sample which has been written over by new data. Then the second INPUT gets a sample from the buffer which is unharmed. (This works because the sample contains enough characters to allow an INPUT to get significantly ahead of the arriving character stream, and because the sample contains fewer characters than the holding buffer.)

Lines 110-130 extract the coordinates from the sample. It was not possible to use an INPUT statement with these number variables because the sample does not have commas between the sample numbers. The details of what the program does with the samples is not shown (in order to keep the example to the important points).

```
10 DIM IN$(16)
20 XIO 36,#1,13,0,"R2:"
30 OPEN #1,5,0,"R2:"
40 XIO 40,#1,0,0,"R2:"
      :
      :
      :
100 INPUT #1,IN$:INPUT #1,IN$
110 LET BUTTON=VAL(IN$(2,2))
120 LET X=VAL(IN$(3,8))
130 LET Y=VAL(IN$(9,14))
      :
      :
      :
590 GO TO 100:REM [Get next point]
      :
      :
      :
```

APPENDIX 10

CODE TABLES

DECIMAL
HEX
ASCII
ATASCII
BAUDOT

DECIMAL CODE	HEX. CODE	ASCII CHARACTER	ATASCII CHARACTER FUNCTION/DISPLAY
0	00	NUL	
1	01	SOH	
2	02	STX	
3	03	ETX	
4	04	EOT	
5	05	ENQ	
6	06	ACK	
7	07	BEL	
8	08	BS	
9	09	HT	
10	0A	LF	
11	0B	VT	
12	0C	FF	
13	0D	CR	
14	0E	SO	
15	0F	SI	
16	10	DLE	
17	11	DC1	
18	12	DC2	
19	13	DC3	
20	14	DC4	
21	15	NAK	
22	16	SYN	
23	17	ETB	
24	18	CAN	
25	19	EM	
26	1A	SUB	
27	1B	ESC	(KEY ESC ESC)
28	1C	FS	MOVE UP ONE LINE =^
29	1D	GS	MOVE DOWN ONE LINE =v
30	1E	RS	MOVE LEFT ONE SPACE =<
31	1F	US	MOVE RIGHT ONE SPACE =>
32	20	SP	SPACE
33	21	!	!
34	22	"	"
35	23	#	#
36	24	$	$
37	25	%	%
38	26	&	&

39	27	'	'
40	28	(	(
41	29	)	)
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[	[
92	5C	\	\

93	5D	]	]
94	5E	^	^
95	5F	_	—
96	60	`	
97	61	a	a
98	62	b	b
99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B	{	
124	7C		
125	7D	}	CLEAR SCREEN =
126	7E	~	BACKSPACE =
127	7F	DEL	TAB 10 SPACES =

Below is a table of the most common Baudot code. All Baudot codes are identical for letters, numbers, and control characters, but they differ somewhat in punctuation. The DECIMAL VALUE column gives the 5-bit Baudot serial binary code converted to decimal. When transmitted, a start bit (space) precedes the character, the character itself is sent low bit first, and 1.5 or 2 stop bits (mark) follow. Mark is sent for 1, space for 0.

The hex/dec columns shwo the value of the Baudot character when interpreted as an 8-ibt word with the three high--order bits set to 1. These are the codes which represent the Baudot characters with the Interface Module's no-translation mode (translation mode 32).

LETTERS	FIGURES	DECIMAL VALUE	HEX/DEC

A	-(dash)	3	E3/227
B	?	25	F9/249
C	:	14	EE/238
D	$	9	E9/233
E	3	1	E1/225
F	!	13	ED/237
G	+ or &	26	FA/250
H	# or STOP	20	F4/244
I	8	6	E6/230
J	' (apost.)	11	EB/235
K	(	15	EF/239
L	)	18	F2/242
M	. (period)	28	FC/252
N	, (comma)	12	EC/236
O	9	24	F8/248
P	0 (zero)	22	F6/246
Q	1 (one)	23	F7/247
R	4	10	EA/234
S	BELL	5	E5/229
T	5	16	F0/240
U	7	7	E7/231
V	;	30	FE/254
W	2	19	F3/243
X	/	29	FD/253
Y	6	21	F5/245
Z	"	17	F1/241
NULL	NULL	0	E0/224
RETURN	RETURN	8	E8/232
LINEFEED	LINEFEED	2	E2/226
SPACE	SPACE	4	E4/228
LETTERS	LETTERS	31	FF/255
FIGURES	FIGURES	27	FB/251

APPENDIX 11

PRINCIPLES OF OPERATION OF THE 850[TM] INTERFACE MODULE

The 850 Interface Module is a computer; it contains a microprocessor, built-in program in ROM, and extensive I/O capability. The I/O forms the parallel (printer) and serial (RS-232-C) ports, and is also used for communication between the Interface module and the ATARI 400 or ATARI 800 computer.

This section presents the theory of operation of the Interface Module. Topics include the automatic bootstrap function, operation of the RS-232-C port handler which is loaded by the bootstrap function into the 400 or 800 computer, operation of the Interface Module to execute the RS-232-C I/O commands, and operation of the printer port. The electrical interfaces of the RS-232-C and printer ports are shown, and signal handshake and timing on the printer port are discussed.

POWER-ON BOOTSTRAPPING OPERATION

NOTE THAT THE BOOTSTRAPPING OPERATION IS REQUIRED ONLY FOR OPERATION OF THE RS-232-C (SERIAL) PORTS AND NOT THE PRINTER PORT. The ATARI 400 and 800 computers already contain the necessary programming to operate the printer port on the 850 Interface Module. The automatic power-on bootstrapping operation, when enabled, loads the 1762-byte handler and tables required for operation of the serial ports.

The bootstrapping operation is enabled by turning ON the power to the Interface Module before the 400 or 800 computer.

Without Disk Drive

When the ATARI Personal Computer System's power is turned on, it issues a disk request. If there is no Disk Drive in the system (or if the Disk Drive is OFF), the Interface Module responds to the disk request. The computer then loads a special bootstrapping program from the Interface Module, as if it were reading from a disk. The bootstrapping program is then run, and it gets the RS-232-C handler from the Interface Module and relocates it into the computer's RAM. The memory occupied by the bootstrapping program is then freed (but the handler remains).

With Disk Drive

If there is a Disk Drive attached to the system (Drive 1 only), it responds to the disk request issued by the 400 or 800 computer at power-on. The computer then reads a start-up program from that disk. Most commonly, this program is an ATARI Disk Operating System (DOS). The Interface Module does not respond to the disk request if a Disk Drive responds first; therefore, the program loaded from the disk must

load the handler from the Interface Module. In the DOS II, this job is handled by a special AUTORUN.SYS file supplied with your DOS II diskette. The AUTORUN.SYS program is loaded and executed by the DOS; it finds the Interface Module and loads the bootstrapping program from it. The bootstrapping program then loads and relocates the RS-232-C handler from the Interface Module. Read the instructions supplied with your DOS II for details on AUTORUN.SYS.

PRINTER SOFTWARE OPERATION

The Interface Module responds to commands to an ATARI printer whenever it senses a printer attached to the parallel port (see the electrical section on the printer port for signal requirements between the Interface Module and a printer).

The ATARI 400/800 Operating System contains a printer handler program which will address one printer, called P: . Four commands are allowed by the P: handler: OPEN, CLOSE, output (represented by PUT, PRINT, and LIST in BASIC), and STATUS.

To use the printer, one must OPEN a channel (IOCB) to the printer. CLOSE releases the channel when it is no longer needed.

ATARI printers (including the Interface Module) operate in Block Output Mode (as described elsewhere in this manual for the RS-232-C port operation) The printer handler builds a 40-byte buffer, and when the buffer fills, the 40 bytes are sent to the printer. When a printer is attached to the Interface Module, the Interface Module accepts the 40 characters and sends them, one at a time, over the parallel port to the attached printer. The printer must acknowledge all 40 characters within 4 seconds (see the electrical section for a discussion of the handshake between the Interface Module and a printer).

There is one exception to the above description: When the printer handler is asked to print an ATASCII EOL (End-of-Line) character, it fills any unused part of the 40-character buffer with blanks (following the EOL) and sends it immediately. For this reason, the Interface Module ignores any characters in the buffer which follow an EOL.

The Interface Module translates EOL into ASCII CR (Carriage Return, 13 decimal). No other translations are made. In particular, bit 7 (high bit) of each byte is not changed, and LF is not added following CR. However, multiple EOL's in a row, without intervening characters, are sent to the printer as alternating CR's and blanks.

A special note about LPRINT in BASIC: LPRINT is equivalent to OPEN, PRINT and CLOSE all in one. Execution of an LPRINT statement with a comma or semicolon at the end will send to the Interface Module a 40-character buffer which is padded with blanks but does NOT have an EOL character The Interface Module will send all 40 characters to the printer (including the blanks), but the printer will probably not

respond because most printers wait for CR before activating a print cycle.

The STATUS request for device P: is answered by the Interface Module if there is a printer attached and powered ON. The status returned in location 746 (decimal) contains 128 if the previous operation to the printer was successful; 129 if the previous command to the Interface Module printer port was bad; 130 if the previous 40-byte data frame had an error (this should not happen); and 132 if the previous command timed out--that is, the printer stayed BUSY past 4 seconds.

RS-232-C PORTS SOFTWARE OPERATION

Once booted, the RS-232-C port handler is linked in as the R: device. This handler contains code to re-establish itself whenever a warm start (RESET) occurs.

The RS-232-C handler is called by CIO to execute each type of I/O operation for the R: device (except output calls from BASIC which bypass CIO by calling the RS-232-C handler directly). Some of the commands are executed entirely by the handler (set-up), but most are passed on to the Interface Module. Some commands cause set-up in both the handler and in the Interface Module.

The CONFIGURE BAUD RATE command is a set-up command which is executed by both the handler and the Interface Module. Both the handler and the Interface module keep separate tables for each of the four RS-232-C ports.

The SET TRANSLATION MODE command is executed by the handler. This command sets values which control the translation and parity handling during I/O.

The CONTROL command is executed by the Interface Module. Outgoing control lines for the indicated port are set ON (or MARK), set OFF (or SPACE), or left alone, as specified by the control parameter. Each line is left alone until another CONTROL command is executed. Note that, if the XMT line is set to SPACE, it will return to SPACE following any subsequent data transmission, until another CONTROL command sets it to MARK.

The OPEN command is executed entirely by the handler. It establishes control information for the port being OPENed. The CLOSE command is executed mostly by the handler: OPEN flags are cleared; any data in output buffers is sent; concurrent mode I/O is shut down. Any data in an input buffer is lost at CLOSE time.

Block mode output takes data from BASIC PRINT or PUT statements, puts each character through translation, and puts each character into the 32-byte output buffer. The buffer is transmitted when it fills, or when 13 (hex) is stored into the buffer (automatic short block on CR). Data from the buffer is sent to the Interface Module as 8-bit bytes. If 7-, 6-, or 5-bit words are configured, the Interface Module strips

the necessary number of high-order bits from each byte before transmitting it to the port. If monitoring of any external status line has been configured for the port, the readiness is checked by the Interface Module whenever a block is sent to it. If not ready, the Interface Module returns a NAK. The 400/800 computer waits while the Interface Module transmits a block.

The FORCE SHORT BLOCK command causes the handler to transmit the block of data before 32 bytes have been collected. If there is no data in the buffer, the FORCE SHORT BLOCK command has no effect.

When START CONCURRENT MODE I/O is performed, a number of things occur. The handler marks the concurrent mode I/O as active (if there are no errors while starting concurrent mode I/O). The handler sets up its own serial input/serial output interrupt handlers as necessary (depending on I/O direction) to field data going in and out. The handler sets itself up to monitor the BREAK key so BREAK will stop the concurrent mode I/O. The handler establishes the initial (empty) state of the input and output buffers. Then the handler informs the Interface Module that concurrent mode I/O has started.

During concurrent mode I/O, each character being received from the Interface Module is taken in by the handler's interrupt driver, put through translation, and placed in the input buffer. Characters to be sent to the Interface Module are translated and put in the output buffer. As the serial hardware in the computer finishes sending each character, the output interrupt driver immediately sends another character from the buffer (unless it is empty). If the input buffer overflows, an error is flagged; output buffer overflow stops putting data into the buffer until data is sent to free buffer space.

Input and output statements (GET, PUT, PRINT, INPUT) executed to a channel through which concurrent I/O is active do not directly cause any I/O to the RS-232-C port. Rather, input statements simply retrieve data that is in the input buffer, and output statements put data into the output buffer. If an input statement wants more data but the input buffer is empty, BASIC will wait until the data arrives. If an output statement attempts to put data into a full output buffer, BASIC will wait until space becomes available (as a result of the interrupt-driven sending of data from the output buffer). The interrupt-driven sending of data from the output buffer starts as soon as data is put into the buffer. The data is moved into and out of each buffer circularly--that is, the buffer is automatically re-used. The maximum amount of data a circular buffer can hold at once is one less byte than its size.

The Interface Module handles concurrent I/O in one of two ways. The most common mode is used when 8-bit words are being transmitted, no matter what the rate of I/O direction. In this mode, the interface module "connects" (through the interface module's microprocessor) the XMT and RCV lines of the selected port to the I/O connector going to the computer. The data is not interpreted by the Interface Module in this mode; all serialization of the data is performed by the serial I/O hardware in the 400/800 computer. Note that the "connection"

between the RS-232-C port and the computer's peripheral I/O port is handled by software. Each line coming in to the Interface Module (one from the computer, one on the RS-232-C port) is sampled (checked) over and over, and its value is then passed on to the "connected" outgoing line. The sampling rate is 34.6 kHz; the lines are sampled every 28.9 microseconds.

The other concurrent I/O mode is established in the Interface Module for low speed (300 Baud or less) 7-, 6-, or 5-bit input (half-duplex). In this mode, the Interface Module receives a 7-, 6-, or 5-bit character from the port and then transmits a corresponding 8-bit character to the computer. This is done because the computer's hardware is not capable of receiving anything but 8-bit serial words. The Interface Module receives the data by sampling it at a rate of 16 samples per bit (similar to a typical UART). As each character is sent from the Interface Module to the computer, extra high-order 1-bits are added to get 8-bit words. The Interface Module sets an internal error flag if a framing error occurs in the incoming data. This flag may be queried with STATUS after the concurrent I/O is stopped.

The Interface Module leaves concurrent mode when told to by the handler when the concurrent I/O channel is CLOSEd, or when BREAK or RESET is pushed.

The Interface Module is constantly keeping track of all incoming RS-232-C readiness lines, for the purpose of being able to report their history to the STATUS command. This does not apply to the RCV lines or any lines on the printer port. The readiness lines are checked periodically through sampling. The sampling rate depends on the activities the Interface Module is asked to perform. In order not to be missed, a pulse on a readiness line should be at least a few dozen milliseconds in duration.

The STATUS command is performed either by the RS-232-C handler alone (when concurrent I/O is active) or by both the handler and the Interface Module. In the former case, the handler supplies the user with information about its current operation. In the latter case, the handler combines some of its own information with status and sense information supplied by the Interface Module.

ELECTRICAL SPECIFICATIONS OF RS-232-C SERIAL PORTS

(You may refer to the schematic diagram of the 850 Interface Module --APPENDIX 12, Figure 6-- while reading this section.)

There are basically two types of circuit for the serial port lines: a receiving circuit, and a transmitting circuit. One of these circuits connects each RS-232-C signal line to a pin of one of the two computer I/O chips in the Interface Module.

The sending circuit consists of an operational amplifier (op-amp) followed by a 10 Ohm protective resistor. The op-amp is driven "to the rail", and produces approximately +9 volts for SPACE, and -5.5

volts for MARK guaranteed at least +or-5V, when driving a 3000 Ohm load (3 k-Ohm is the worst-case load allowed by the RS-232-C standard; any lower resistance may result in improper operation). The driver circuit will withstand short circuits to ground, and will withstand connection to voltages within their driving range. Shorting a driver to a voltage outside the range -5.5 to +9 volts may result in damage to the Interface Module.

The receiving circuit consists of a diode and transistor whose function is to convert the minus/plus RS-232-C voltages to the voltages used by the I/O chips. A 4700 Ohm input resistor protects the outside device from having to deliver too much current. Notice that the DSR inputs have 1800 Ohm resistors attached to ground which insure that DSR will seem OFF if nothing is attached to DSR (however, this is no protection against the "antenna effect" of a long unterminated wire attached to DSR, which will cause DSR to go ON and OFF if there is activity in other leads in the same cable).

Port 4 may be set up for 20 mA current loop operation. In current-loop operation, pins 4 and 7 (RTS +10v, and RCV) are tied together (Pin numbers are of the 9-pin connector of the Interface Module). When the attached Teletype keyboard-sending contacts are closed, pin 9 pulls RCV negative (MARK). This is the idle state of the Teletype. Whenever the switch opens during transmission of a character from the Teletype, RCV is pulled positive (SPACE). Notice that if the Teletype if turned off this switch may be open and the 850 Interface Module will receive a BREAK signal.

For current loop output, the Teletype's printer solenoid is tied between pins 1 and 3 (+10v DTR and XMT). XMT is normally negative (MARK); thus the solenoid is activated in the MARK state. XMT goes to nearly +10v for SPACE so very little current passes through the solenoid and it disengages. Be careful when connecting a current loop device that it does not apply excessive voltages to the Interface Module. Also note that if the send and receive loops are connected together within the Teletype the send or receive loop may not work correctly (the signal may be shorted out). If this happens, try swapping the send or receive wire pairs.

PARALLEL PRINTER PORT SPECIFICATIONS

While reading this section you may refer to the Interface Module schematic --APPENDIX 11, Figure 6-- and printer port timing diagrams at the end of this section.

All signals on the printer port are TTL level (0 to +5v). The output lines are buffered by transistors to supply the necessary drive for the printer electronics. Input lines are buffered to protect the I/O chips.

The output circuit can sink 5 milliAmps. That is, the circuit is capable of pulling 1000 Ohm pull-up resistors in the printer to TTL zero. The output circuit expects some such pull-up in the printer; if

they are not present, the output lines will be pulled to +5v only by the internal 10000 Ohm pull-up resistors and the lines may slew too slowly to TTL one.

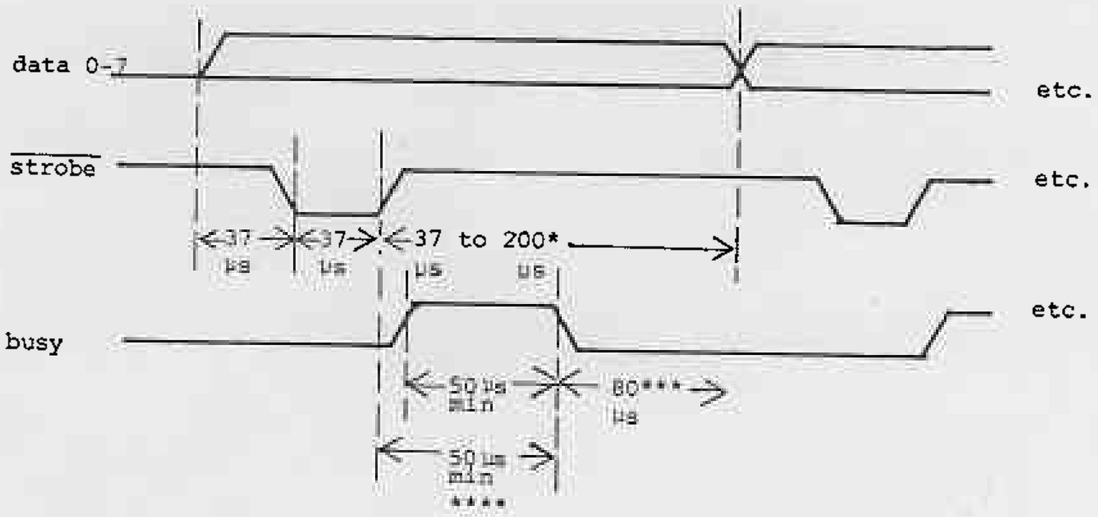
The Interface Module detects the presence of the printer via the FAULT line. If this line is low, the Interface Module will not respond to printer requests from the 400 or 800 computer. This line is low if the 825 printer is turned off (or disconnected). This feature allows you to connect more than one ATARI printer to the computer I/O port, and switch between them by powering only one of them on at a time. If you attach your own printer to the printer port, FAULT must be high (TTL one) for the Interface Module to operate the printer. If there is no appropriate signal from your printer to which FAULT may be attached, you may connect FAULT (pin 12) to the +5 volt pull-up at pin 9. Be sure you do not connect FAULT to a busy-type line which will alternate on and off; FAULT should stay on.

The eight data lines are positive-logic. The data lines normally rest at zero (ASCII NULL). A data byte is sent to the printer (when it is not BUSY) by placing the data on the 8 data lines and pulsing the DATA STROBE. The STROBE is normally high, and goes low during the strobe pulse.

After sending each data byte to the printer, the Interface Module waits for a BUSY signal. The ATARI 825 printer sends a positive-logic BUSY signal as it processes each byte of data. The BUSY is quite short for most data bytes since the printer merely saves each character in its own memory, but BUSY is quite long when the printer prints. The Interface Module does not care how long the printer is BUSY -- the only requirement is that the printer respond to all 40 characters (that is, go not BUSY after the last character) within 4 seconds. Immediately after BUSY goes low again, the Interface Module sends the next character to the printer. When all the characters have been accepted by the printer, the Interface Module signals the 400/800 computer that the print operation is finished.

Some printers which use the Centronix-type interface do not signal BUSY for each character received, but only go BUSY during printing. For this reason, the Interface Module only waits 200 microseconds for BUSY after sending a data byte. If BUSY does not go on within this time, the Interface Module sends the next character, assuming the printer has completed its processing of the preceding character.

Figure 1 Timing Of Printer Port



* one byte sent every 280 µs without Busy (see text)

** pulse must be > 50µs, no maximum. However, 40 characters must be accepted by printer in 4 sec. (see text)

*** approximate

**** Busy may follow either leading or trailing edge of strobe, however, it must remain at least 50 µs after trailing edge of strobe.

Figure 6. Printer Port timing diagram.

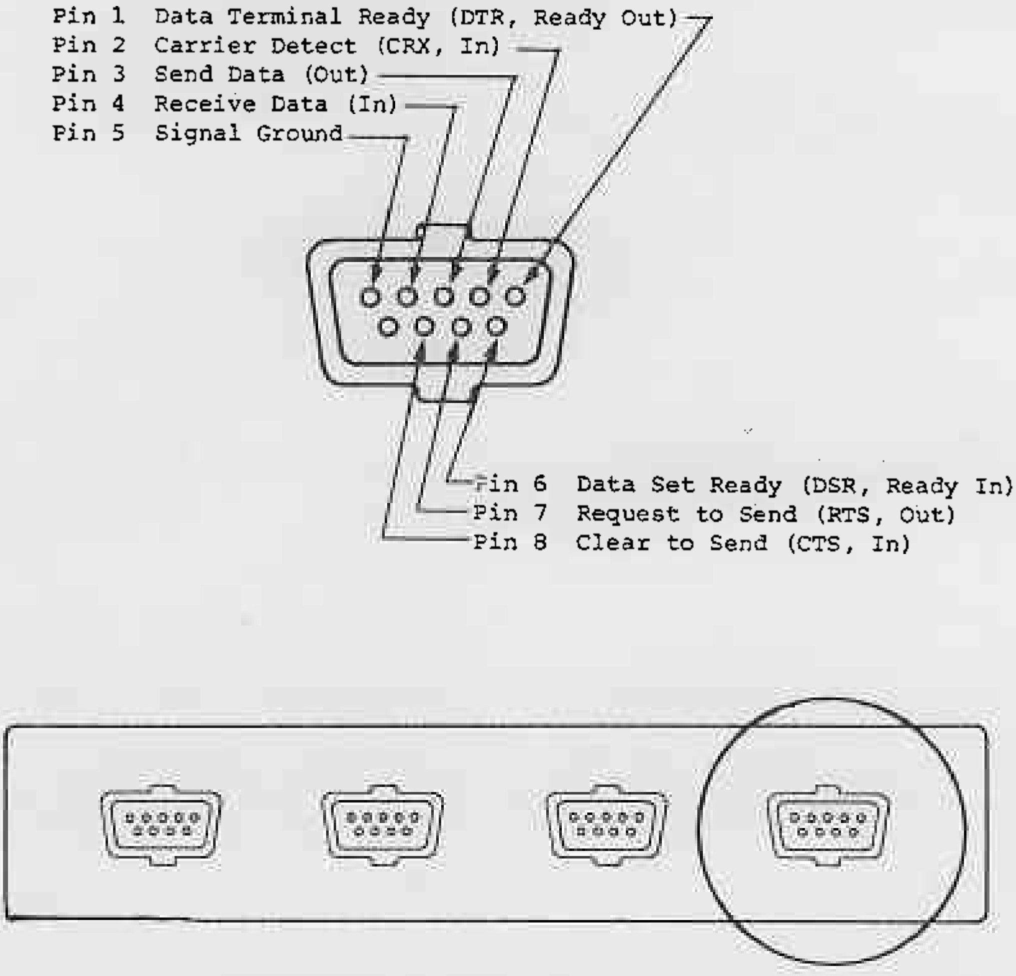


Figure 1. Pin functions of Serial Port No. 1 in 850™ Interface Module (9-pin female connector)

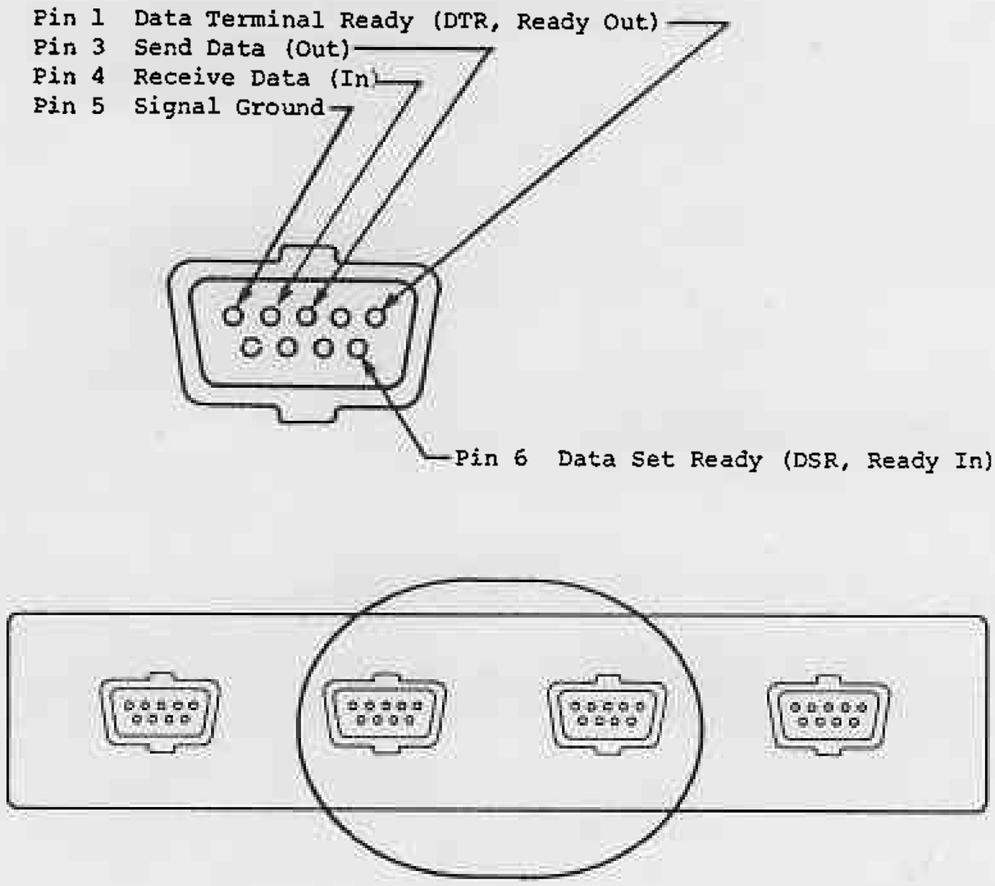


Figure 2. Pin functions of Serial Port Nos. 2 and 3 in 850™ Interface Module (9-pin female connector)

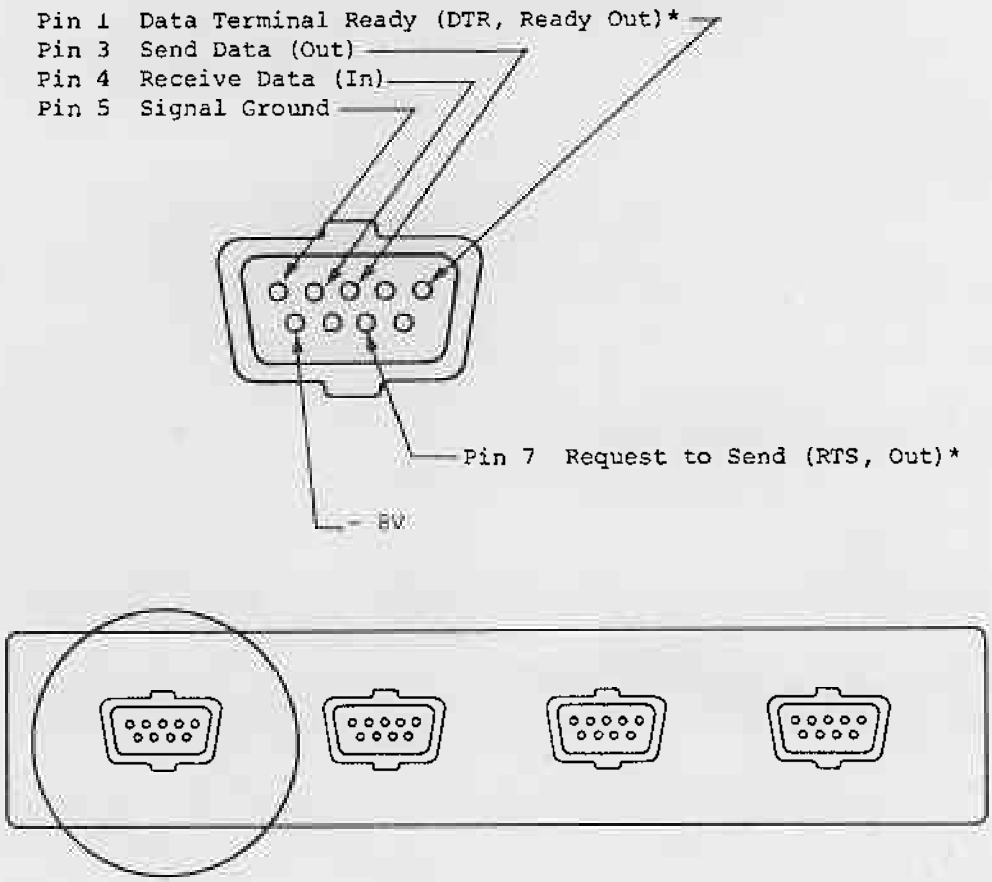


Figure 3. Pin functions of Serial Port No. 4 in 850™ Interface Module (9-pin female connector)

*These pins are not computer-controlled and are always ON (+10v).

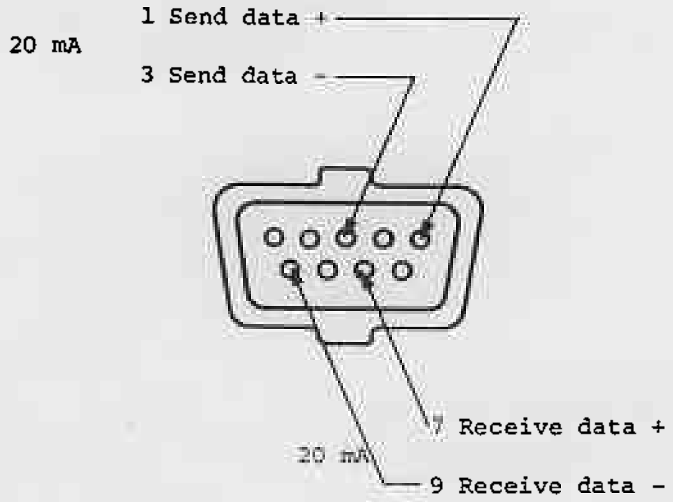


Figure 4. Hook-up of Serial Port No. 4 used with a 20 mA loop device.

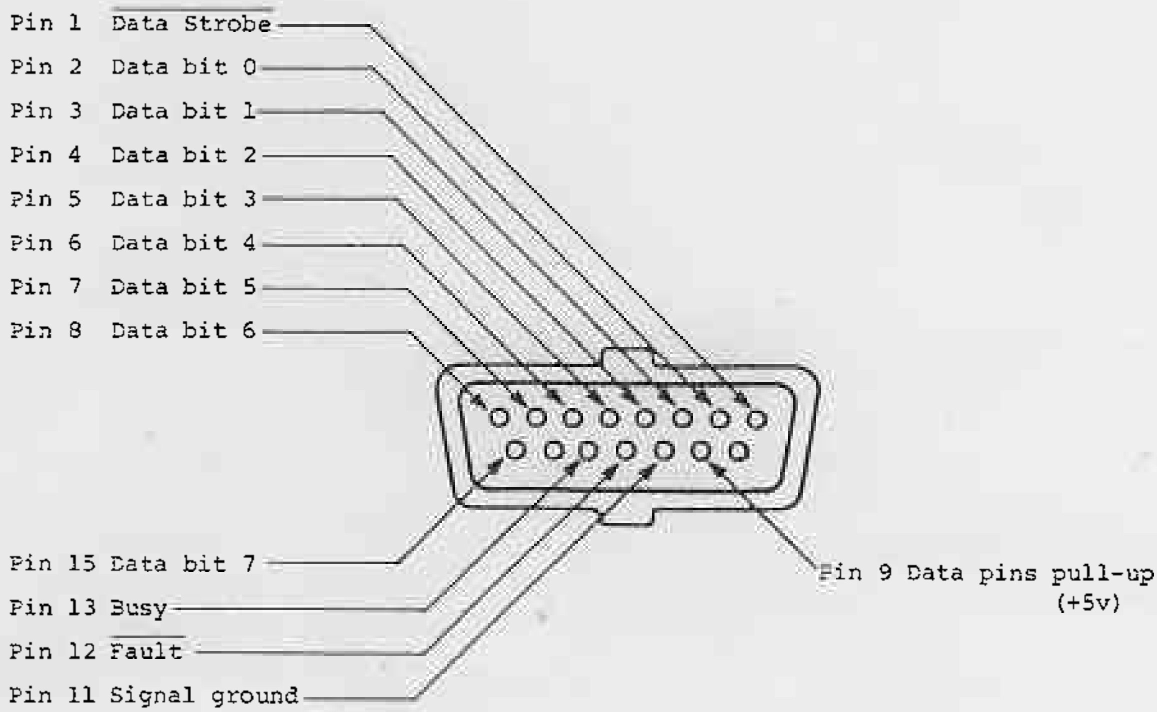


Figure 5. Pin functions of the Printer Port of the 850™ Interface Module (15-pin female connector

Note 1: This port is designed for connecting the ATARI 825™ Printer. If you are going to connect a different printer, consult the specifications of its input connector. Make sure that the cable you use to connect between the Printer Port of the 850 and the input to the printer is wired correctly to connect Bit 0 to Bit 0, and so on. You may have to construct your own cable.

Note 2: $\overline{\text{Fault}}$ must be +5 for printer port to operate. If your printer has nothing appropriate to connect to FAULT (such as +5 power or some ONLINE-type signal), connect Pin 9 to Pin 12.

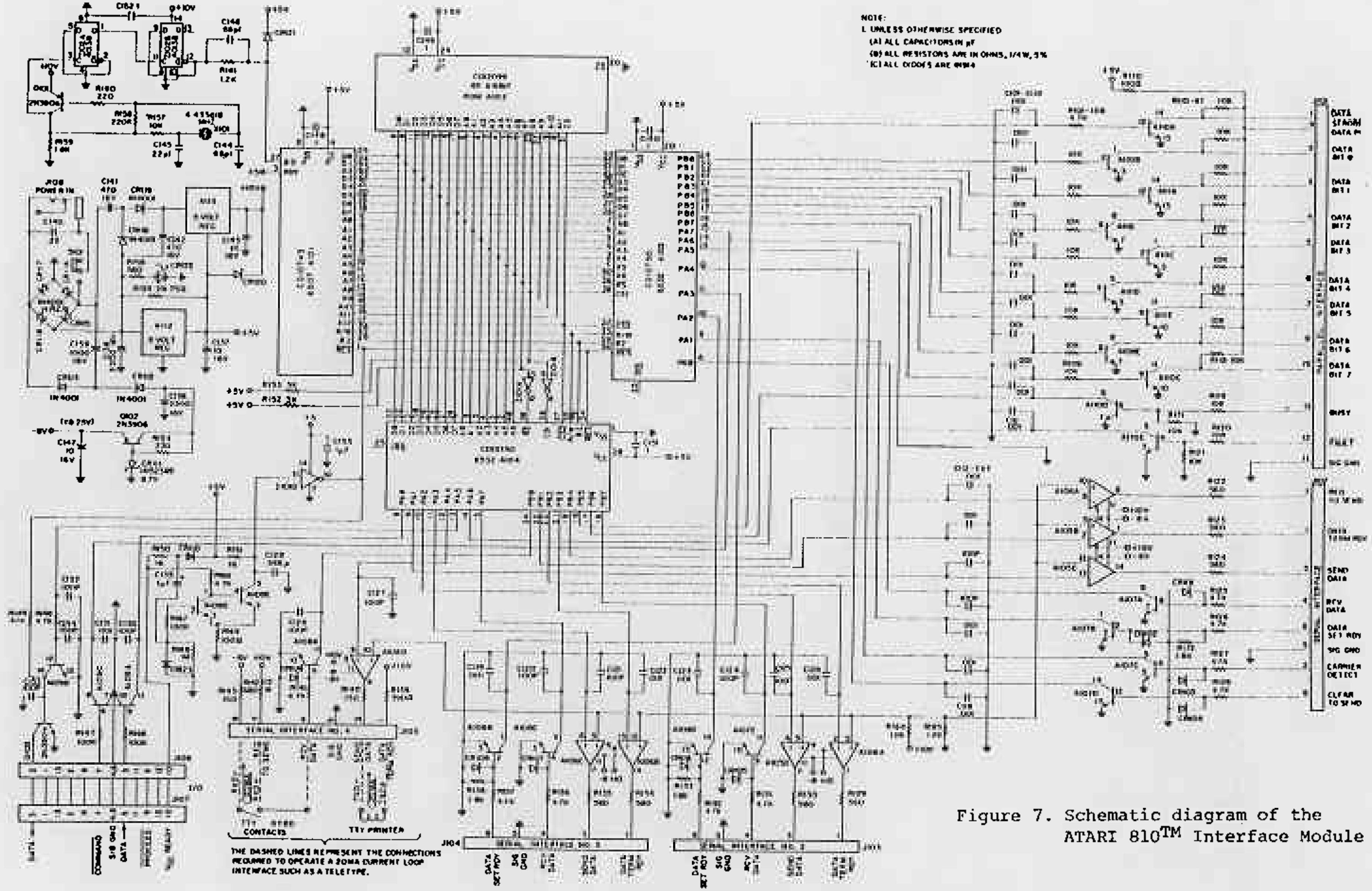


Figure 7. Schematic diagram of the ATARI 810TM Interface Module

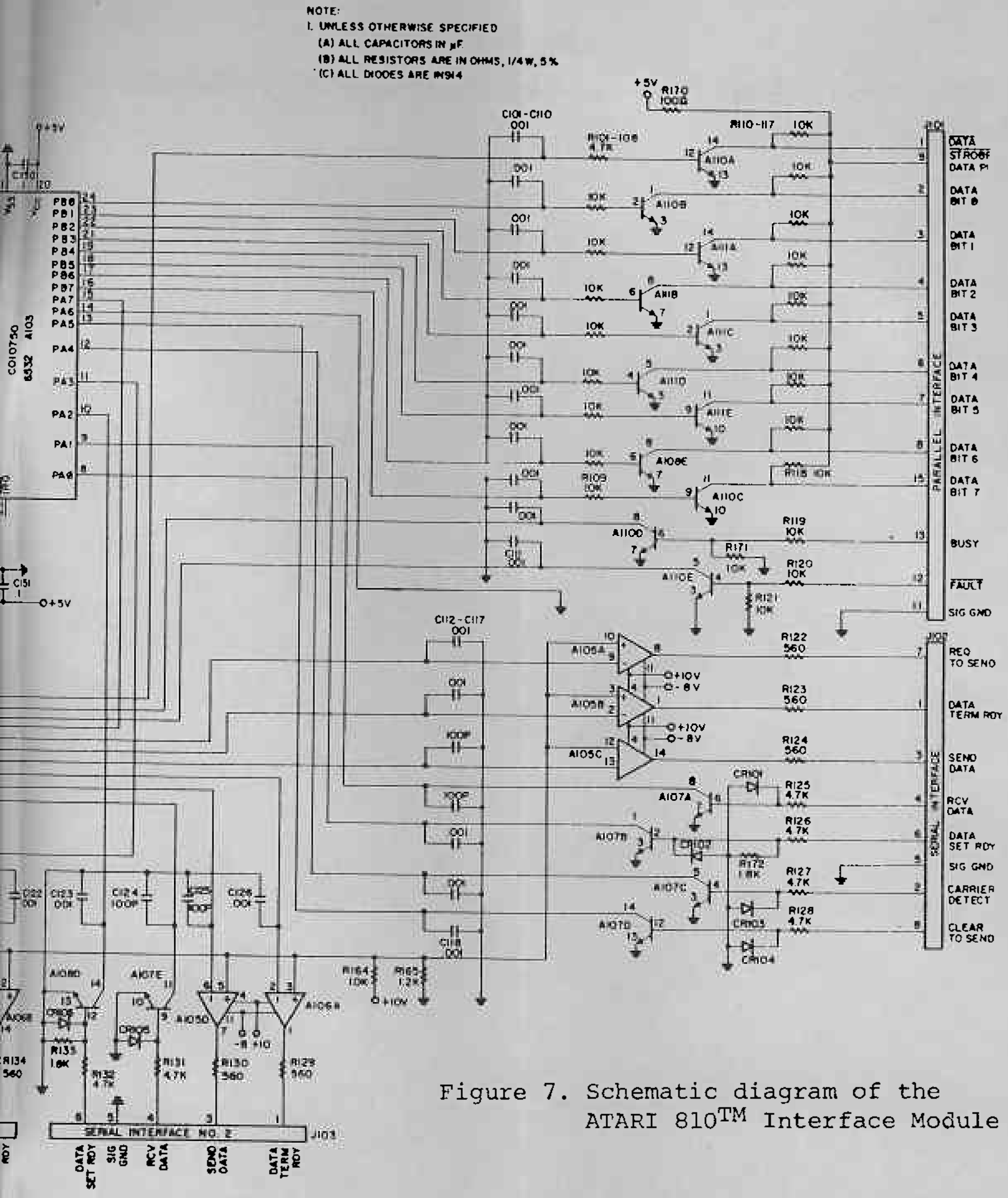
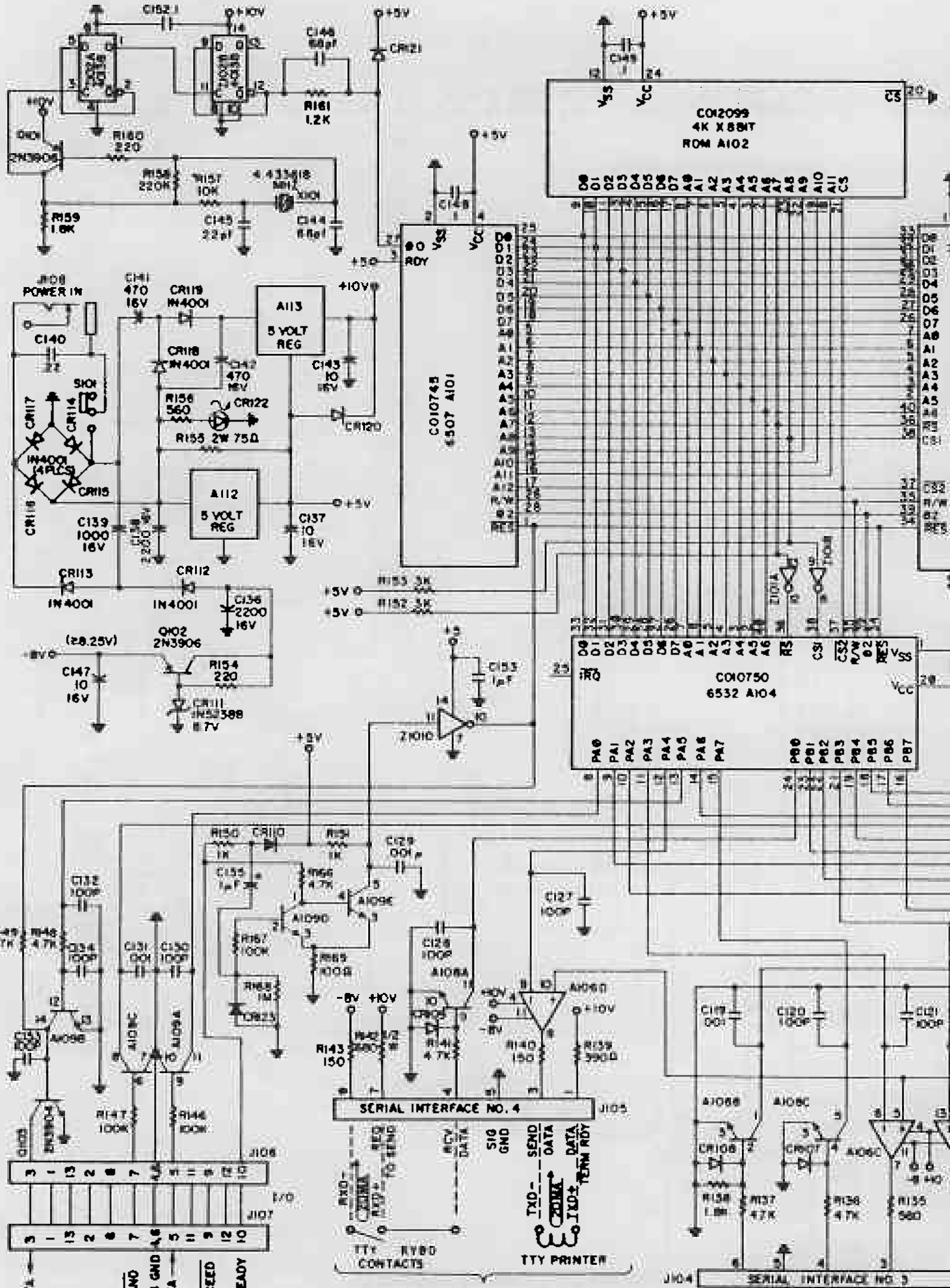


Figure 7. Schematic diagram of the ATARI 810TM Interface Module



THE DASHED LINES REPRESENT THE CONNECTIONS REQUIRED TO OPERATE A 20MA CURRENT LOOP INTERFACE SUCH AS A TELETYPE.

APPENDIX 13

ATARI 830[tm] Modem

The term "modem" is a contraction of "modulator/demodulator". In this context, modulate means to convert from serial binary data to signals of particular frequencies and demodulate means to convert from signals of particular frequencies to serial binary data. The ATARI 830 is a modem. It allows you to send and receive data over telephone lines at rates up to 300 Baud (equivalent to 30 characters/second).

To use the modem you need an ATARI 400 or 800, the ATARI 850 Interface Module and a telephone. Hook-up is shown on the packing box of the modem.

The modem uses frequencies in the audio range. The communication paths include small volumes of air between the modem and the ear-piece and mouthpiece of the telephone. Hence, a modem of this type is said to be "acoustically coupled".

The common standard for communications between computer-related equipment is the RS-232-C standard of the Electronic Industries Association (see APPENDIX 1). The ATARI 830 modem conforms with that standard. The most common type of Modem in use is the Bell 103/113 series. The ATARI 830 modem can communicate with Bell 103/113 equipment.

SIZE

10.2 x 4.7 x 2.3 inches

WEIGHT

1.5 lbs.

TEMPERATURE

Operating environment 32 to 122 degrees Fahrenheit. (0 to 50 degrees Centigrade)

Storage: -40 to 140 degrees Fahrenheit. (-40 to 60 degrees Centigrade)

ELECTRICAL REQUIREMENTS

24 VAC/150mA supplied by UL-listed wall-mount transformer with 6 foot cord.

HUMIDITY

Operating Environment: 10% to 90% relative humidity (no condensation).

Storage: 5% to 95% (no condensation).

TRANSMITTER FREQUENCIES

Originate:

Mark:	1270 Hz
Space:	1070 Hz

Answer:

Mark:	2225 Hz
Space:	2025 Hz

RECEIVE FREQUENCIES

Originate:

Mark:	2225 Hz
Space:	2025 Hz

Answer:

Mark:	1270 Hz
Space:	1070 Hz

MAXIMUM TRANSMISSION RATE

300 Baud

RECEIVE SENSITIVITY

-45dBm

CONTROLS

FULL/TEST/HALF

FULL: Sets full duplex operation.
TEST: Sets up audio self-test mode.
HALF: Sets half duplex operation.

Receive data will copy transmit data.

ANS/OFF/ORIG

ANS: Sets answer mode.
OFF: Turns unit power off.
ORIG: Sets originate mode.

INDICATORS

POWER: Power on.
READY: Ready to communicate.

DATA INTERFACE

The modem provides an RS-232-C interface via a standard 25 pin female D-connector. The table below lists the signals used by the modem.

OUTPUTS:	Mark(OFF):	-8V
	Space(ON):	+10V
INPUTS:	Mark(OFF):	-3 to -25V
	Space(ON):	+3 to +25V

PIN CONNECTIONS

PIN #	SIGNAL MNEMONICS	FUNCTION	SIGNAL DIRECTION
2	XMT	Transmit Data	Input to Modem
3	RCV	Receive Data	Output to terminal
5	CTS	Clear to Send (On with Carrier Detect)	Output to terminal
6	DSR	Data Set Ready (ON with Carrier Detect)	Output to terminal

7		Signal Ground	Common
8	CRX	Carrier Detect	Output to terminal

INSTALLATION

Set modem power off, connect wall-mount transformer to unit and 115 VAC outlet.

Connect terminal RS-232-CC cable to interface connector on modem.

Turn modem to ORIG or ANS. POWER indicator should be ON.

OPERATION--
ORIGINATE MODE

Set ANS/OFF/ORIG control to ORIG. and FULL/TEST/HALF control as required.

Dial remote number. After hearing answering tone, place handset into acoustic muffs with cord as indicated on label.

When READY indicator turns on, the modem is ready to communicate.

Proceed with communication.

OPERATION--ANSWER MODE

Manually answer telephone, switch ANS/OFF/ORIG control to ANS.

NOTE: In answer mode a tone should be heard at all times with or without the READY indicator being activated.

Place handset into muffs with cord as indicated by label.

When READY indicator turns on, proceed with normal data exchange.

To terminate call, switch back to OFF and hang up telephone.

TEST

This test mode is desgined to verify that the modem is functioning properly. It does so by switching the transmitter channel frequencies to match those of the receiver. All data into the modem will be looped back to the terminal for verification. It requires a telephone set to provide an isolated acoustic path between speaker (transmitter) and microphone (receiver).

ORIGINATE MODE
----

1. When communicating with a time share computer the modem should be set to ORIG mode. Modem at remote computer end will be in answer mode.
2. When commmunicating with another terminal, mode selection is determined by prior agreement between users. Remember one modem must be in answer mode, the other in originate mode.
3. For proper communication, modem must be either FULL or HALF, not in TEST.
4. Is modem at the other end com: patible with the modem. Remote modem must be another modem or a Bell 103-compatible modem. Communication cannot be established with a Bell 202 type.

Double Character Display:

Is Modem in half duplex mode?

1. If remote computer echoes all characters both modem and your terminal should be in FULL duplex mode.
2. If communication system is half duplex (no echo), either your terminal or the modem (not both) should be in HALF duplex.

Garbled Display:

Is telephone handset fully seated in the rubber muffs?
Is baud rate correct?
Both local and remote terminals must send data at the same baud rate (300 baud or less).
Is received signal too weak or noisy?
Pick-up handset and listen for a clean tone (if remote modem is in answer mode). If additional tones, dialing pulses, static noise or voices are present, data may be garbled. Re-dial call.

Set terminal for full duplex operation and Modem to ORIG. Set FULL/TEST/HALF switch to TEST. A continuous tone should be heard from the speaker muff. If no tone is present, unit is defective.

Dial a single digit on the telephone to obtain a quiet line situation. Immediately place handset into acoustic muffs with cord as indicated on label.

NOTE: A quiet line is required for this test to prevent dial tone interference. The line # may remain quiet for only 30 seconds. Repeat as necessary. A longer quiet time can be obtained by dialing a telephone extension or another number under control. The mouthpiece of the second telephone set must be covered to prevent room noise interference.

Wait for READY indication, type message on keyboard. The TEST function will display message. Check terminal for message accuracy.

ANSWER--MODE

Once Modem passes test in originate mode, quickly switch to answer mode with telephone handset still in the rubber muffs.

Wait for READY indication, type message on keyboard. The TEST function will display message. Check terminal for message accuracy.

NOTE: If Modem gives READY indication in both answer and originate modes yet no message or incorrect message (such as double characters) is displayed on the terminal, the RS-232-C cable or terminal interface may be at fault. If terminal can be looped back with RS-232-C cable (pin 2 tied to pin 3 at modem end) and correct message is displayed, then the modem is defective.

TROUBLE SHOOTING

If you have problems, the most likely reason is the phone line. Noise on the line or a weak phone line signal can often result in lost or invalid data. Try to re-dial the call to insure connection is noise-free and there is no interference.

If communication still cannot be established and modem checks out in the TEST mode, see tables below for other possible causes for failure.

Ready Light Off:	Is modem power ON? Is handset in proper position? Label indicates direction of cord. Are mode switches set properly?

======
NOTICE
======

Use of multiple printer control codes that involve carriage motion (with the exception of end-of-line), can cause an ERROR 139 (Device NAK). Carriage motion includes backspace, forward and reverse linefeeds, and partial linefeeds.

The ATARI 850[tm] Interface Module sends data to the printer in 40-character blocks. If there is more than one carriage motion in each block, the printer cannot recover in time to receive the next 40-character block.

If you should have this problem, check your program. Try to arrange your printer control codes in such a way that there is no more than one carriage motion in each 40-character block. This can be done by preceding each carriage motion with forty "null" characters. Null characters can be generated with control comma ([CTRL] [,]) or with CHR$(0).